KeySafe 5

# KeySafe 5 v1.0 Installation Guide

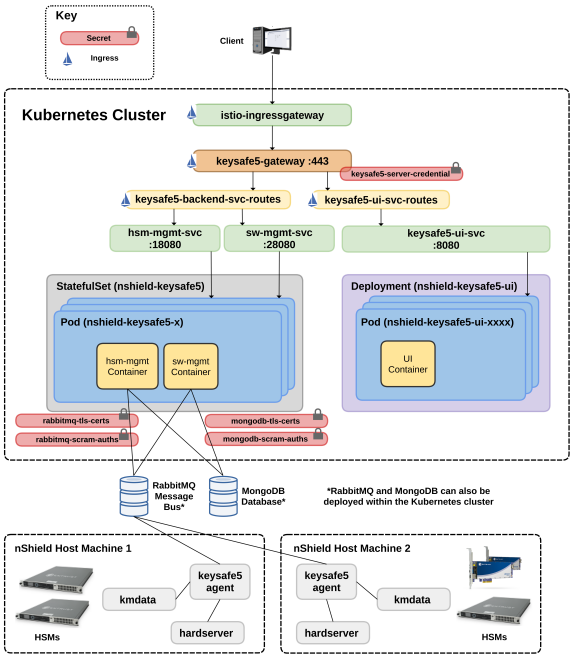
8 April 2024

# Table of Contents

KeySafe 5 provides a centralised means to securely manage a distributed nShield HSM estate, including the management and creation of Security Worlds and associated resources (Softcards & Card Sets).

KeySafe 5 provides this capability in two forms: HTTP REST APIs for HSM Management and Security World management, and a graphical user interface. Only authenticated clients are permitted access to the service, providing assurance that your HSM and Security World data remain usable only by clients that are permitted access.

Typical KeySafe 5 deployment:




The main central management platform of KeySafe 5 is deployed as a Kubernetes application. For each nShield client machine that you want to manage using this platform, you must install a KeySafe 5 agent binary alongside the existing nShield hardserver.

# 1. Release Package

The release package is provided in `.tar.gz` format and has the following contents

## 1.1. OpenAPI specifications

The API specification documents for the RESTful web services follow v3 of the OpenAPI specification.

- `api/hsm-mgmt.yml` defines the HSM Management API
- `api/sw-mgmt.yml` defines the Security World Management API

## 1.2. Helm charts

The KeySafe 5 Kubernetes-based deployment consists of 3 Helm charts:

- `helm-charts/nshield-keysafe5-backend-1.0.0.tgz`

  This installs the backend API services (HSM Management and Security World Management).

- `helm-charts/nshield-keysafe5-ui-1.0.0.tgz`

  This installs the graphical user interface for KeySafe 5.

- `helm-charts/nshield-keysafe5-istio-1.0.0.tgz`

  This configures an existing Istio Ingress Gateway to allow external access (routing and authentication) to the services deployed by the other two Helm charts.

This split enables you to deploy the backend services only, if you do not need the UI, or the UI only, if you want to point it at some existing backend services already running elsewhere.

You can also use a different Kubernetes Ingress other than Istio if desired.

For more information on configuring and installing the Helm chart, see Helm Chart Installation.

## 1.3. Docker images

The Docker images are provided as tar archives. You can load them into a local Docker image registry using the `docker load` command, then push to a private container registry

For example:

```
$ docker load < docker-images/hsm-mgmt.tar
Loaded image: hsm-mgmt:1.0.0
$ docker tag hsm-mgmt:1.0.0 private.registry.local/keysafe5/hsm-mgmt:1.0.0
$ docker login private.registry.local
$ docker push private.registry.local/keysafe5/hsm-mgmt:1.0.0
```

The Docker images provided are:

- `docker-images/hsm-mgmt.tar` is the HSM Management service
- `docker-images/sw-mgmt.tar` is the Security World Management service
- `docker-images/ui.tar` is the KeySafe 5 user interface

These Docker images are intended to be deployed via the provided Helm charts. See the Helm chart configuration for details of how to configure and run each image.

## 1.4. KeySafe 5 agent installers

You can use the Linux and Windows installers provided to install the KeySafe 5 agent on nShield client machines. See KeySafe 5 Agent Installation for details on configuring and installing the agent.

# 2. Prerequisites

The following table contains the required software version that this release of KeySafe 5 has been tested on and any minimum version requirements.

In addition to the set of required software, this release of KeySafe 5 requires an external identity provider that supports OIDC and OAuth2 for user and machine authentication.

| Software | Minimum version | Tested version |
|---|---|---|
| Kubernetes | - | 1.22 |
| MongoDB | 4.0 | 4.4.12 |
| RabbitMQ | 3.0 | 3.9.13 |

## 2.1. Optional Software

KeySafe 5 is shipped with a Helm chart that configures an Istio Ingress Gateway to provide external access to the KeySafe 5 application running in Kubernetes. Other Ingress Gateways can be used, see Helm Chart Installation: Configure a customer ingress provider.

| Software | Minimum version | Tested version |
|---|---|---|
| Istio | - | 1.13 |

## 2.2. Kubernetes cluster

KeySafe 5 has been tested on Kubernetes version 1.22

### 2.2.1. Using namespaces

When deploying an application to a Kubernetes cluster that is shared with many users spread across multiple teams, Entrust recommends using Namespaces to isolate groups of resources.

To create a namespace for the KeySafe 5 application:

```
$ kubectl create namespace nshieldkeysafe5
namespace/nshieldkeysafe5 created
```

To set the namespace for a current request, use the `--namespace` flag. For example:

```
$ helm install --namespace=nshieldkeysafe5 my-release helm-keysafe5-backend/
$ kubectl --namespace=nshieldkeysafe5 get pods
```

If you are using Istio in your Kubernetes cluster, beyond acting as an API Gateway for KeySafe 5, you might also want to configure Istio injection for this Kubernetes namespace to take advantage of other Istio features. This step is not required for KeySafe 5 to function.

```
$ kubectl label namespace nshieldkeysafe5 istio-injection=enabled
namespace/nshieldkeysafe5 labeled
```

## 2.3. MongoDB

MongoDB is the persistent data store for the KeySafe 5 application data. Any sensitive Security World data stored in the database is stored in standard nShield encrypted blobs. You should restrict access to this database in the same way that you would normally restrict access to the `Key Management Data` directory on an nShield client machine.

The MongoDB used must be a Replica Set.

If you have an existing MongoDB database you can configure the application to use this, otherwise you must securely deploy a MongoDB instance.

Entrust recommend that you configure MongoDB with authentication enabled and TLS enabled and with RBAC configured. The database user for KeySafe 5 should be given the minimum capabilities required, see Database: User Roles.

### 2.3.1. Bitnami MongoDB Helm chart

To install MongoDB in the same Kubernetes cluster as the KeySafe 5 application, you can install the Bitnami MongoDB Helm chart

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm install mongo-chart \
 --set image.tag=4.4.12 \
 --set architecture=replicaset \
 --set auth.enabled=true \
 --set tls.enabled=true \
 --set tls.existingSecret=my-mongo-tls-secret \
bitnami/mongodb --version 11.1.10
```

KeySafe 5 has been tested with version 11.1.10 of the Bitnami MongoDB Helm Chart.

Monitor MongoDB for security vulnerabilities and regularly update the version of MongoDB installed to apply any required updates.

## 2.4. RabbitMQ

You need a RabbitMQ message bus for the interprocess communication between the KeySafe 5 backend services and keysafe5-agent instances running on nShield client machines.

If you have an existing RabbitMQ instance you can configure the application to use this, otherwise you must securely deploy a RabbitMQ instance.

Entrust recommend that you configure RabbitMQ with authentication enabled and TLS enabled. If the RabbitMQ instance being used is shared with other applications, then a dedicated virtual host should be created for the KeySafe 5 application. The RabbitMQ user used for KeySafe 5 should only have access to that specific virtual host.

### 2.4.1. Bitnami RabbitMQ Helm chart

To install RabbitMQ in the same Kubernetes cluster as the KeySafe 5 application, you can install the Bitnami RabbitMQ Helm chart.

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm install rabbit-chart \
 --set image.tag=3.9.13 \
 --set auth.username=user \
 --set auth.existingPasswordSecret=my-password-secret \
 --set auth.tls.enabled=true \
 --set auth.tls.existingSecret=my-rabbitmq-tls-secret \
bitnami/rabbitmq --version 8.32.1
```

KeySafe 5 has been tested with version 8.32.1 of the Bitnami RabbitMQ Helm Chart.

Monitor RabbitMQ for security vulnerabilities and regularly update the version of RabbitMQ installed to apply any required updates.

## 2.5. External identity provider (IdP)

You need an external identity provider that supports OIDC and OAuth2 to provide user and machine authentication to KeySafe 5. This is required to gain access to the UI and authenticate commands sent to the backend services.

At the IdP, Entrust typically expects the following to be configured:

- A single OIDC public client application

  This provides user identity information and an `id_token` for use by the UI

- Multiple OAuth2 private client application

  This provides machine-to-machine credentials. Typically you would want an instance of this per application identity required to limit the sharing of the `client_secret` value.

For more information, refer to Client Types.

### 2.5.1. OIDC public client

The OIDC public client application provides user identity information and an `id_token` for use by the UI in its calls to the backend services. It is a public client due to the UI being a client side application, and as such cannot be trusted with the `client_secret` like a server side application would be.

Entrust recommends the following settings:

| Setting | Value |
|---|---|
| Grant Types | Authorization Code with PKCE |
| | Refresh Token |
| Authorization Code PKCE Code Challenge Method | S256 |
| Scopes | openid |

For more information, refer to Authorization Code flow with PKCE.

### 2.5.2. OAuth2 private client

The OAUTH2 private client provides client credentialing for machine-to-machine

authentication by applications that do not hold user identification. This requires the use of the `client_secret` value, which must be securely held.

You would typically want a separate private client instance for each application for which you provide access, resulting in a separate `client_id` and `client_secret` for each application. This eases management of the `client_secret` by reducing the number of application that have knowledge of it. It also provides easy identification of which application is doing what at the KeySafe 5 end.

Entrust recommend the following settings:

| Setting | Value |
| --- | --- |
| Grant Type | Client Credentials |

For more information, refer to Client Credentials.

# 3. Helm Chart Installation

To install the KeySafe 5 REST APIs you must install the helm-keysafe5-backend chart.

To install the KeySafe 5 graphical user interface you must also install the helm-keysafe5-ui chart.

To expose these services externally to the Kubernetes cluster, see Configure external access to KeySafe 5.

## 3.1. Helm

Helm is a package manager for Kubernetes.

As with `kubectl` you can apply a namespace to the Kubernetes resources created by a Helm chart by specifying `--namespace my-namespace` when using `helm`.

To install the chart with the release name `my-release`:

```
$ helm install my-release helm-keysafe5-backend/
```

List all releases using `helm list`.

To upgrade or modify the existing `my-release` deployment, configure the chart parameters and then run:

```
$ helm upgrade --install my-release helm-keysafe5-backend/
```

To uninstall or delete the `my-release` deployment:

```
$ helm delete my-release
```

## 3.2. helm-keysafe5-backend

The keysafe5-backend Helm chart deploys Kubernetes Services for the KeySafe 5 REST APIs.

- hsm-mgmt (nShield HSM Management Service API)

- sw-mgmt (nShield Security World Management Service API)

This Helm chart installs the KeySafe 5 services into a Kubernetes cluster but does not configure external access to the services. See Configure external access to KeySafe 5

### 3.2.1. Kubernetes Secrets

The helm-keysafe5-backend Helm chart expects to be provided with pre-existing Kubernetes Secrets for the Database and AMQP connections.

For more information, refer to Kubernetes Secrets.

| Purpose | Description | Secret Type |
|---|---|---|
| MongoDB SCRAM Auth | The username and password pairing used to authenticate with the MongoDB server | kubernetes.io/basic-auth |
| MongoDB TLS Certificates | TLS certificates used to connect to the MongoDB server. Can also be used as authentication if X509 auth is enabled | Opaque |
| AMQP SCRAM Auth | The username and password pairing used to authenticate with the AMQP server | kubernetes.io/basic-auth |
| AMQP TLS Certificates | TLS certificates used to connect to the AMQP server | Opaque |

Because TLS secrets have the `Opaque` type, the filenames they are created with are critical. Entrust expects the following:

- CA certificate to be named `ca.crt`.
- TLS Certificate to be named `tls.crt`.
- TLS key file to be named `tls.key`.

Your certificates will need to adhere to X.509 v3, sometimes known as a multiple-domain certificates, or SAN certificates. The X.509 extension Subject Alternative Name (SAN) allows specifying multiple hostnames, and has replaced Common Name as the source of the hostname.

### 3.2.2. Configuration

To deploy the application, configure the chart with:

- The Docker images to use for `sw-mgmt` and `hsm-mgmt`.
- Database connection configuration.
- AMQP connection configuration.

For further details on the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart.

An example install:

```
$ helm install my-release \
  --create-namespace --namespace nshieldkeysafe5 \
  --set database.type=mongo \
  --set database.mongo.hosts="mongo-chart-mongodb-0.mongo-chart-mongodb-
headless.mongons.svc.cluster.local:27017,mongo-chart-mongodb-1.mongo-chart-mongodb-
headless.mongons.svc.cluster.local:27017" \
  --set database.mongo.replicaSet=rs0 \
  --set database.mongo.auth.type=pwd \
  --set database.mongo.auth.existingSecret=my-mongo-credentials-secret \
  --set database.mongo.tls.enabled=true \
  --set database.mongo.tls.existingSecret=my-mongo-tls-secret \
  --set amqp.URL="rabbit-chart-rabbitmq-0.rabbit-chart-rabbitmq-headless.rabbitns.svc.cluster.local:5671" \
  --set amqp.auth.type=pwd \
  --set amqp.auth.existingSecret=my-amqp-credentials-secret \
  --set amqp.tls.enabled=true \
  --set amqp.tls.existingSecret=my-amqp-tls-secret \
  --set logging.level=info \
  --set logging.format=json \
 helm-keysafe5-backend/
```

## 3.3. helm-keysafe5-ui

The keysafe5-ui Helm chart deploys a Kubernetes Service for the KeySafe 5 Graphical User Interface.

The chart deploys the web front-end of KeySafe 5. For the UI to be usable it must point to a deployed KeySafe 5 back-end services endpoint (as installed by helm-keysafe5-backend).

This Helm chart installs the KeySafe 5 UI into a Kubernetes cluster, but does not configure external access to the service. (See Configure external access to KeySafe 5.)

### 3.3.1. Configuration

To deploy the application, the configure the chart with:

- The Docker image to use for the `ui` container.

- OIDC Identity Provider config, if Authentication is enabled.
- The location of the KeySafe 5 back-end services API that this UI displays information for.

For further details on the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart.

An example install:

```
# Untar the chart and copy your OIDC provider config file into the config directory
$ tar -xf helm-charts/nshield-keysafe5-ui-1.0.0.tgz -C helm-charts
$ cp my-oidc-provider-config.json helm-charts/nshield-keysafe5-ui/config/OIDCProviders.json

# Install the chart
$ helm install keysafe5-ui \
  --create-namespace --namespace nshieldkeysafe5 \
  --set svcEndpoint="https://XXX.XXX.XXX.XXX" \
  --set authMethod=oidc \
  --wait --timeout 10m \
helm-charts/nshield-keysafe5-ui-1.0.0.tgz
```

Because the OIDC Provider configuration is volume mapped into the Kubernetes application, you must untar the packaged Helm chart so that you can copy in the `OIDCProviders.json` file to the correct location before installing the chart.

### 3.3.2. Configure UI authentication

If `authMethod` is set to `oidc` then you must provide an OIDC configuration file detailing the accepted Identity Providers.

Copy the OIDC configuration file to `config/OIDCProviders.json` in the root directory of the Helm chart before installing the Helm chart. This file is a JSON document. You can find an example file at `config/OIDCProviders.json.example`.

The configuration document is a JSON list of individual provider configurations.

An example of provider configuration:

```
{
    "name":"Example Provider",
    "authority": "https://example-auth-provider.com/auth/auth",
    "client_id":"8acc1449-7275-4524-b25f-4a60ddddfe8d",
    "redirect_uri": "https://example-keysafe5.com/callback",
    "response_type": "code",
    "scope": "openid",
    "post_logout_redirect_uri": "https://example-keysafe5.com/",
    "issuer": "https://example-auth-provider.com/auth",
    "authorization_endpoint": "https://example-auth-provider.com/auth/callback/authorize",
    "token_endpoint": "https://example-auth-provider.com/auth/token",
```

```
    "jwks_uri": "https://example-auth-provider.com/auth/keys",
    "end_session_endpoint": "https://example-auth-provider.com/auth/logout",
    "userinfo_endpoint": "https://example-auth-provider.com/auth/userinfo",
    "iconSVG" : "login.svg"
}
```

To display a custom icon for the provider in the user interface, copy an SVG-format image to the `config` directory and reference the file name under the key `iconSVG` in the provider configuration.

## 3.4. Configure external access to KeySafe 5

To enable external access to the services installed by `helm-keysafe5-backend` and `helm-keysafe5-ui`, you need to configure a Kubernetes Ingress Gateway to route requests to the appropriate Kubernetes Services.

If you use Istio, you can use the `helm-keysafe5-istio` Helm chart to configure an existing Istio Ingress Gateway. Alternatively, you can configure your own Ingress Gateway. (See Configure a customer ingress provider.)

### 3.4.1. helm-keysafe5-istio

The keysafe5-istio Helm chart creates an Istio Gateway and VirtualService for the KeySafe 5 back-end services and user interface to be accessible externally from the Kubernetes cluster.

The chart:

- Routes HTTP requests to the KeySafe 5 application running in the same Kubernetes cluster.
- Authenticates requests, if authentication is enabled (default) and an Identity Provider (IdP) is configured.
- Applies CORS policy to requests to limit Cross-Origin Resource Sharing.
- Applies security related HTTP Headers to responses including automatically applying the Content-Security-Policy header to help reduce Cross Site Scripting (XSS) risks.
- Limits TLS protocol used to TLS v1.2 and higher.
- Limits ciphers supported by the gateway.

### 3.4.2. Configure helm-keysafe5-istio

For further details of the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart.

An example install:

```
$ helm install keysafe5-istio \
  --create-namespace --namespace=nshieldkeysafe5 \
  --set requireAuthn=true \
  --set httpsEnabled=true \
  --set portNumber=443 \
  --set issuer[0].authIssuer="https://foobar.auth0.com" \
  --set issuer[0].authJWKsURI="https://www.googleapis.com/oauth2/v1/certs" \
  --set issuer[0].authAudiences[0]="https://keysafe5.location" \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.0.0.tgz
```

### 3.4.3. helm-keysafe5-istio port number

If the port number changes to something other than 443 or 80, you need to open a port on the Istio Ingress Gateway. You can do this using your own istioctl install manifest. For more information, refer to https://istio.io/latest/docs/setup/install/istioctl/.

### 3.4.4. helm-keysafe5-istio authentication

Authentication can be provided by any OpenID Connect Provider, such as Entrust Identity As A Service.

If `requireAuthn` is `true`, then at least one `authIssuer` must be configured.

An example configuration:

```
issuers:
  - authIssuer: 'https://foobar.auth0.com'
    authJWKsURI: 'https://www.googleapis.com/oauth2/v1/certs'
    authJWKs: ''
    authAudiences:
    - 'https://keysafe5.location'
```

For each `authIssuer`, `authAudiences` and one of `authJWKsURI` or `authJWKs` must be specified

| Key | Description |
|---|---|
| `authIssuer` | Identifies the issuer that issued the JWT. A JWT with different iss claim will be rejected. |

| Key | Description |
| --- | --- |
| `authAudiences` | Identifies the list of JWT audiences that are allowed access. A JWT containing any of these audiences will be accepted. |
| `authJWKsURI` | URL of the provider's public key set to validate signature of the JWT. For more information, refer to https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata. |
| `authJWKs` | JSON Web Key Set of public keys to validate signature of the JWT. For more information, refer to https://auth0.com/docs/jwks |

For additional information, refer to https://istio.io/latest/docs/reference/config/security/jwt/#JWTRule

Once authenticated, a user has full access to view and manage the HSM and Security World resources accessible from the platform. No authorization policy is applied to requests.

### 3.4.5. Enable HTTPS for helm-keysafe5-istio

Configuring a TLS certificate for the Istio Gateway requires creating a Kubernetes secret.

To create a Kubernetes secret from an existing TLS private key and certificate:

You must create the Kubernetes Secret in the same namespace as the Istio Ingress Gateway

```
$ kubectl --namespace istio-system create secret tls keysafe5-server-credential \
  --cert=path/to/cert/file \
  --key=path/to/key/file
```

## 3.5. Configure a customer ingress provider

If you do not want to use Istio, you can configure your own Kubernetes Ingress.

If you configure your own Ingress to the application, then it is your responsibility to configure routing to the services and any authentication or authorization to access the services.

Each part of the KeySafe 5 application is exposed as a Kubernetes Service.

`helm-keysafe5-backend` exposes the following Kubernetes Services for serving requests to the RESTful APIs:

| ClusterIP Port | API endpoints |
| --- | --- |
| 18080 | /mgmt/v1/hsms<br><br>/mgmt/v1/pools |
| 28080 | /mgmt/v1/worlds |

`helm-keysafe5-ui` exposes a Kubernetes Service called `keysafe5-ui-svc` on CluserIP port `8080` for accessing the graphical user interface of KeySafe 5.

# 4. KeySafe 5 Agent Installation

The KeySafe 5 agent runs alongside the existing hardserver and enables the central management platform to manage all HSMs and Security Worlds visible to the hardserver.

The KeySafe 5 agent is a privileged client of the hardserver. For more information on privileged clients, see the nShield Security World Software documentation.

The connection between the agent and the central monitoring platform is via the RabbitMQ message bus using the Advanced Message Queueing Protocol (AMQP). It is configured in the KeySafe 5 agent configuration file.

The KeySafe 5 agent ensures that all key management data, with the exception of keys, is synchronised between the nShield client machine and a central (MongoDB) database.

This means that when resources, such as Card Sets or Softcards, appear in the `kmdata/local` directory on a client machine, they are automatically stored in the central database. It also means that when a Card Set or Softcard is created via the new management tools, the resource also appears in `kmdata/local` on any host machine that is in the right Security World.

The Card Set or Softcard can then be used with the traditional nShield tools on each nShield client machine.

If a resource is deleted via the KeySafe 5 application then it will be removed from `kmdata/local` for all client machines running a KeySafe 5 agent. If the resource is deleted locally on a nShield client machine then that deletion is not synchronised to other client machines in the same Security World.

## 4.1. Configuration

The KeySafe 5 agent configuration file is located at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml`. The install contains an example configuration file at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml.example`.

Unless configured otherwise, `%NFAST_DATA_HOME%` is located at `/opt/nfast` on Linux and `C:\ProgramData\nCipher` on Windows.

| Configuration Key | Description | Example Value |
|---|---|---|
| `override_hostname` | Set the hostname for this agent. This appears as the Pool name in KeySafe 5. The hostname is set by the operating system if not overriden here. This is not related to the hostname used by TLS authentication. | `hostname` |
| `logging.level` | Minimum severity level of log statements to output. Valid values: `trace`, `debug`, `info`, `warning`, `error`. The default is to output at `info` level and above. | `info` |
| `logging.format` | Format of the log statements. Valid values: `json`, `logfmt`. The default is to output in `json` format. | `json` |
| `logging.file.enabled` | To enable log output to file, set to `true`. The default is to output to file (`true`). | `true` |
| `logging.file.path` | The absolute path of the file that logs should be written to The default is `/opt/nfast/log/keysafe5-agent.log` on Linux and `C:\ProgramData\nCipher\Log Files\KeySafe5-agent.log` on Windows. | `/opt/nfast/log/keysafe5-agent.log` |
| `amqp.auth_type` | Authentication method for the AMQP connection. Valid values: `none`, `pwd`, `tls`. The default is to use TLS authentication. | `tls` |
| `amqp.url` | The URL points to the AMQP service (IP address or DNS name) along with its port number. This parameter is required, there is no default. | `127.0.0.1:5671` |
| `amqp.disable_tls` | To disable Mutual TLS for the AMQP connection, set to `true`. The default is to use Mutual TLS (`false`). | `false` |
| `update_interval` | The period of time between publishing data updates. The interval string is a sequence of decimal numbers, each with optional fraction and a unit suffix, such as "300ms", "1.5h" or "2h45m". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h". The default is once a minute. | `1m` |
| `health_interval` | The period of time between checking the underlying service health and attempting recovery if necessary. The format is as used for `update_interval`. The default is once a minute. | `1m` |

| Configuration Key | Description | Example Value |
|---|---|---|
| `kmdata_poll_interval` | The rate at which the agent polls the `kmdata` directory to look for changes. The format is as used for `update_interval` The default is once a second. | `1s` |
| `amqp.min_protocol_version` | The minimum tls protocol version that is used by the AMQP connection of the keysafe5 agent. The default is `TLSV1_2`. | `TLSV1_2` |
| `amqp.cipherSuites` | The available ciphersuites for the AMQP connection of the keysafe5 agent. The default is `ECDHE-ECDSA-AES128-GCM-SHA256, ECDHE-RSA-AES128-GCM-SHA256, ECDHE-ECDSA-AES256-GCM-SHA384, ECDHE-RSA-AES256-GCM-SHA384, ECDHE-ECDSA-CHACHA20-POLY1305, ECDHE-RSA-CHACHA20-POLY1305` | `ECDHE-ECDSA-AES128-GCM-SHA256` |

Configure the KeySafe 5 agent's AMQP connection to use the same RabbitMQ instance used by the central management platform that you want to connect to.

## 4.2. AMQP authentication

You can configure the authentication method for the AMQP connection as one of the following options:

- `none` No authentication.
- `pwd` Username and password authentication.
- `tls` x.509 certificate authentication.

Entrust recommends restricting access to files containing sensitive authentication details. On Linux, the KeySafe 5 agent installer will create the required files with appropriate permissions.

### 4.2.1. Password authentication

To configure password authentication, set `amqp.auth_type` to `pwd` in the configuration file. Locate the plaintext username and password in the following files on disk:

- `%NFAST_DATA_HOME%/keysafe5/conf/amqp/auth/username` should contain the username to use for the AMQP connection.

- `%NFAST_DATA_HOME%/keysafe5/conf/amqp/auth/password` should contain the password to use for the AMQP connection.

### 4.2.2. TLS

The TLS key and certificates for the agent's connection to AMQP should be stored within `%NFAST_DATA_HOME%/keysafe5/conf/amqp/tls` in the following files:

- `ca.crt` - The CA Certificate.
- `tls.key` - The agent's private key.
- `tls.crt` - A valid certificate of the key signed by the Certificate Authority.

Your certificates will need to adhere to X.509 v3, sometimes known as a multiple-domain certificates, or SAN certificates. The X.509 extension Subject Alternative Name (SAN) allows specifying multiple hostnames, and has replaced Common Name as the source of the hostname.

## 4.3. Installing on Linux

1. Untar the KeySafe 5 agent install package to the root directory of the machine.

   This unpacks the agent and associated scripts into the `/opt/nfast/` directory.

   ```
   $ cd /
   $ sudo tar -xf /path/to/keysafe5-1.0.0-Linux-keysafe5-agent.tar.gz
   ```

2. Configure this KeySafe 5 agent instance as described earlier in the chapter.
3. Run the nShield install script.

   This will restart the hardserver. The agent must point to a working RabbitMQ otherwise it will fail to start.

   ```
   sudo /opt/nfast/sbin/install
   ```

   The installer creates the following items, as required:

   - Either a SysV-style init script or systemd script for automatically starting and stopping the service.
   - The `keysafe5d` user.

This user is dedicated to running the `keysafe5-agent` service.

## 4.4. Installing on Windows

The hardserver TCP ports must be enabled. To do so, run `config-serverstartup.exe --port 9000 --privport 9001` or by editing the file (located at `%NFAST_KMDATA%\config\config`) and setting `nonpriv_port=9000` and `priv_port=9001`.

After enabling the hardserver TCP ports, you must restart the hardserver service.

If those ports are not available and different ports are set, then the environment variables `NFAST_SERVER_PORT` and `NFAST_SERVER_PRIVPORT` must also be set appropriately as mentioned in the nShield documentation. They may be set globally in System Environment Variables, or only for this service by adding a `Multi-String Value` named `Environment` under `Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nShield KeySafe 5 Agent`, and to *Value data* adding the lines `NFAST_SERVER_PORT=port-number` and `NFAST_SERVER_PRIVPORT=port-number`. You may need to restart the computer after adding the System Environment Variables.

1. Launch the `Keysafe5-agent.msi` installer.

   This creates the nShield KeySafe 5 agent service but does not start it.

2. Populate the KeySafe 5 agent configuration file as detailed previously.
3. Start the nShield KeySafe 5 agent service using Windows Service Manager.

# 5. Uninstall

## 5.1. Central platform

To fully remove the KeySafe 5 application from your Kubernetes cluster, use `helm uninstall`. This uninstalls all KeySafe 5 Helm charts.

```
$ helm list --short --all-namespaces
keysafe5-backend
keysafe5-istio
keysafe5-ui

$ helm uninstall keysafe5-backend --namespace="nshieldkeysafe5"
release "keysafe5-backend" uninstalled

$ helm uninstall keysafe5-ui --namespace="nshieldkeysafe5"
release "keysafe5-ui" uninstalled

$ helm uninstall keysafe5-istio --namespace="nshieldkeysafe5"
release "keysafe5-istio" uninstalled
```

KeySafe 5 application data remains in your MongoDB database after uninstalling the application. To clear this data from the database, remove the databases defined by `hsm_mgmt.dbName` and `sw_mgmt.dbName` in the `helm-keysafe5-backend` chart.

## 5.2. KeySafe 5 agent

Before uninstalling the nShield KeySafe 5 agent, Entrust recommends that you back up any configuration files and certificates from the install.

### 5.2.1. Linux

To remove the KeySafe 5 agent from a Linux host, first stop the KeySafe 5 agent if it is currently running:

```
/opt/nfast/scripts/init.d/keysafe5-agent stop
```

Then run the following command:

```
/opt/nfast/scripts/uninstall.d/keysafe5-agent
```

Then proceed to remove the following files and directories:

- `/opt/nfast/keysafe5`
- `/opt/nfast/sbin/keysafe5-agent`
- `/opt/nfast/lib/versions/keysafe5-agent-atv.txt`
- `/opt/nfast/scripts/install.d/65keysafe5-agent`
- `/opt/nfast/log/keysafe5-agent.log`
- `/opt/nfast/log/keysafe5-agent.pid`

The agent log file will be located in a different location if you have changed the default value of `logging.file.path` in the agent configuration file.

If required, you can also remove the `keysafe5d` user that was created as part of the installation.

### 5.2.2. Windows

To remove the KeySafe 5 agent from a Windows host:

1. Open the **Control Panel** and select **Programs and Features**.
2. Select the **nShield KeySafe 5 Agent** package.
3. Select **Uninstall** and follow the on-screen instructions.

To remove any configuration files, delete the `%NFAST_DATA_HOME%\keysafe5` directory and remove the log file located at `C:\ProgramData\nCipher\Log Files\KeySafe5-agent.log`

The agent log file will be located in a different location if you have changed the default value of `logging.file.path` in the agent configuration file.

# 6. Security Guidance

Your nShield HSM protects the confidentiality and integrity of your Security World keys. However, as with all secure systems, administrators must remain diligent concerning the entities who are given access the system. All network traffic between KeySafe 5 and clients using the UI, the REST API, or both passes through a secure channel. This TLS based secure channel is set up using token-based client authentication.

Entrust recommends the following security-related actions:

- Ensure that log levels are set appropriately for your environment.

  More verbose log levels might expose information that is useful for auditing users of KeySafe 5, but the log information also reveals which REST API operations were performed. While this log information might be useful for diagnostics, it could also be considered sensitive and should be suitably protected when stored.

- Rotate the logs regularly. The log files could grow quickly if left unattended for a long time. The system administrator is responsible for log rotation.
- Verify the integrity of the KeySafe 5 tar file before executing it. You can verify the integrity of this file with the published hash.
- Suitably protect the network environment of KeySafe 5 to maintain its availability, for example using firewalls and intrusion detection and prevention systems.
- Ensure that the KeySafe 5 platform's system clock is set accurately and only authorized system administrators can modify it so that the platform correctly interprets certificate and token lifetimes.
- Ensure that only authorized system administrators have access to the KeySafe 5 system, and only trusted software is run on the platform hosting KeySafe 5.
- Take standard virus prevention and detection measures on the platform hosting KeySafe 5.
- The system administrator should consider whether threats in the KeySafe 5 deployment environment would justify the encryption of the sensitive configuration data held in Kubernetes secrets, see Kubernetes documentation.

# 7. Quick Start Guide

The following steps provide a quick-start guide to installing KeySafe 5 and its dependencies into an existing Kubernetes cluster.

These steps install KeySafe 5 and its dependencies. They should be followed to set up a demo environment for evaluation purposes and should *not* be used for production environments.

For a production deployment you must perform the following actions:

- Secure the connections to MongoDB and RabbitMQ
- Require authentication to access KeySafe 5
- Configure HTTPS for the KeySafe 5 endpoints
- Use your secure CA for certificates

## 7.1. Unpack the release

```
$ mkdir keysafe5-install
$ tar -xf nshield-keysafe5-1.0.0.tar.gz -C keysafe5-install
$ cd keysafe5-install

# Load the Docker images to your local Docker
$ docker load < docker-images/hsm-mgmt.tar
$ docker load < docker-images/sw-mgmt.tar
$ docker load < docker-images/ui.tar

# We need to use a private registry in many places
$ export DOCKER=private.registry.local

# Tag the Docker images for a private registry
$ docker tag hsm-mgmt:1.0.0 $DOCKER/keysafe5/hsm-mgmt:1.0.0
$ docker tag sw-mgmt:1.0.0 $DOCKER/keysafe5/sw-mgmt:1.0.0
$ docker tag mgmt-ui:1.0.0 $DOCKER/keysafe5/mgmt-ui:1.0.0

# Log in to ensure pushes succeed
$ docker login $DOCKER

# And push
$ docker push $DOCKER/keysafe5/hsm-mgmt:1.0.0
$ docker push $DOCKER/keysafe5/sw-mgmt:1.0.0
$ docker push $DOCKER/keysafe5/mgmt-ui:1.0.0
```

## 7.2. Set up a CA

You should use your existing CA for a production system. This is simply used an example for the purposes of having a working demo system.

The CA we will create is based on OpenSSL 1.1.1 - and is run inside a directory of your choosing, but in the examples here we will use `/home/user/keysafe5-install/demoCA`. In that directory, create the file demoCA.conf with the contents:

```
[ ca ]
default_ca      = CA_default                          # The default ca section

[ CA_default ]

dir             = /home/user/keysafe5-install/demoCA  # The directory of the CA
database        = $dir/index.txt                      # index file.
new_certs_dir   = $dir/newcerts                       # new certs dir

certificate     = $dir/cacert.pem                     # The CA cert
serial          = $dir/serial                         # serial no file
#rand_serial    = yes                                 # for random serial#'s
private_key     = $dir/private/cakey.pem              # CA private key
RANDFILE        = $dir/private/.rand                  # random number file

default_days    = 15                                  # how long to certify for
default_crl_days= 5                                   # how long before next CRL
default_md      = sha256                              # Message Digest
policy          = test_root_ca_policy
x509_extensions = certificate_extensions
unique_subject  = no
# This copy_extensions setting should not be used in a production system.
# It is simply used to simplify the demo system.
copy_extensions = copy

[ test_root_ca_policy ]
commonName = supplied
stateOrProvinceName = optional
countryName = optional
emailAddress = optional
organizationName = optional
organizationalUnitName = optional
domainComponent = optional

[ certificate_extensions ]
basicConstraints = CA:false

[ req ]
default_bits       = 4096
default_md         = sha256
prompt             = yes
distinguished_name = root_ca_distinguished_name
x509_extensions    = root_ca_extensions

[ root_ca_distinguished_name ]
commonName = hostname

[ root_ca_extensions ]
basicConstraints       = CA:true
keyUsage               = keyCertSign, cRLSign
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical,CA:true
```

Remember to update the `dir` value to the directory in which the demoCA.conf and the other CA files will be stored.

To generate the long-term CA key and random number source, we first make a directory called `private`:

```
$ mkdir ~/keysafe5-install/demoCA/private
```

Then run:

```
$ openssl genrsa -out ~/keysafe5-install/demoCA/private/cakey.pem 4096
$ openssl rand -out ~/keysafe5-install/demoCA/private/.rand 1024
```

The CA needs a self-signed certificate; as this is a short-term demo it will be valid for 6 months:

```
$ openssl req -x509 -new -nodes \
  -key demoCA/private/cakey.pem \
  -subj "/CN=demoCA" -days 183 \
  -out demoCA/cacert.pem \
  -config demoCA/demoCA.conf
$ cp demoCA/cacert.pem ca.crt
```

And finally, to finish off the configuration:

```
$ mkdir demoCA/newcerts
$ echo 01 > demoCA/serial
$ touch demoCA/index.txt
```

## 7.3. Install and set up the supporting software

### 7.3.1. Istio

```
$ istioctl install -y
```

### 7.3.2. RabbitMQ

We recommend that you use your standard secure RabbitMQ installation, along with your policies for authentication and virtual hosts on your production system; this is only a demo system.

First, we need to generate the TLS keys, and guest password. We need to add the network addresses through which RabbitMQ will be accessed to the certificate, and are very dependent on the configuration of the Kubernetes cluster.

```
$ openssl genrsa -out ~/keysafe5-install/rabbit.key 4096
$ export DNS1="*.rabbit-chart-rabbitmq-headless.rabbitns.svc.cluster.local"
$ export DNS2=rabbit-chart-rabbitmq.rabbitns.svc
$ export DNS3=rabbitmq.rabbitns.svc.cluster.local
$ export DNS4=host.docker.internal
$ export LOCALIP=127.0.0.1
$ export HOSTIP=$(hostname -I | cut -f1 -d" ")
$ openssl req -new -key ~/keysafe5-install/rabbit.key \
  -out ~/keysafe5-install/rabbitmq.csr -subj \
  "/CN=rabbitmq/C=GB/L=Cambridge" \
  -addext "keyUsage=digitalSignature" \
  -addext "extendedKeyUsage=serverAuth" \
  -addext
"subjectAltName=DNS:rabbitmq,DNS:${DNS1},DNS:${DNS2},DNS:${DNS3},DNS:${DNS4},DNS:${HOSTNAME},IP:${LOCALIP},IP:${H
OSTIP}"
$ openssl ca -config ~/keysafe5-install/demoCA/demoCA.conf \
  -out rabbit.crt \
  -in rabbitmq.csr -batch
$ rm rabbitmq.csr
$ kubectl create namespace rabbitns
$ kubectl create secret generic rabbitmq-certificates \
  --namespace=rabbitns \
  --from-file=ca.crt \
  --from-file=tls.crt=rabbit.crt \
  --from-file=tls.key=rabbit.key
$ kubectl -n rabbitns create secret generic rabbitmq-pw \
  --from-literal=rabbitmq-password=guest
```

Then install RabbitMQ.

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami && helm repo update
$ helm install rabbit-chart \
  --set image.tag=3.9.13 \
  --set auth.username=guest \
  --set auth.existingPasswordSecret=rabbitmq-pw \
  --set auth.tls.enabled=true \
  --set auth.tls.existingSecret=rabbitmq-certificates \
  --set extraConfiguration='
    listeners.ssl.default = 5671
    ssl_options.versions.1 = tlsv1.3
    ssl_options.depth = 0
    ssl_options.verify = verify_peer
    auth_mechanisms.1 = EXTERNAL
    ssl_cert_login_from = subject_alternative_name
    ssl_cert_login_san_type = dns
    ssl_cert_login_san_index = 0' \
  --wait --timeout 5m --namespace=rabbitns bitnami/rabbitmq --version 8.32.1
```

You will be given a url that may be used from within the cluster. As we will use the same cluster to install KeySafe 5 we keep that stored in a variable to be used later. We have added `${HOSTNAME}` to the DNS as an externally accessible address for use by the KeySafe 5 agent later.

```
$ export RABBIT_URL=rabbit-chart-rabbitmq.rabbitns.svc:5671
```

Add the virtual host that will be used for KeySafe 5 communication.

```
$ export RUN_RABBIT="kubectl -n rabbitns exec rabbit-chart-rabbitmq-0 -c rabbitmq -- "
$ export RABBIT_VHOST=nshieldvhost
$ ${RUN_RABBIT} rabbitmqctl add_vhost ${RABBIT_VHOST}
```

Then add the x509 users for KeySafe 5, enable x509 authentication, then make RabbitMQ accessible from outside the cluster. It's also a good idea to set up the x509 keys and certificates for KeySafe 5 and its agents.

```
$ export AGENT_USER=keysafe5-agent
$ export KS5_USER=ks5
$ for x509user in $AGENT_USER $KS5_USER
  do
    ${RUN_RABBIT} rabbitmqctl add_user $x509user "ephemeralpw"
    ${RUN_RABBIT} rabbitmqctl set_permissions -p $RABBIT_VHOST $x509user ".*" ".*" ".*"
    ${RUN_RABBIT} rabbitmqctl clear_password $x509user
    openssl genrsa -out $x509user.key 4096
    openssl req -new -key $x509user.key -out $x509user.csr \
      -subj "/CN=${x509user}/C=GB/L=Cambridge" \
      -addext "keyUsage=digitalSignature" \
      -addext "extendedKeyUsage=clientAuth" \
      -addext "subjectAltName=DNS:${x509user}"
    openssl ca -config ~/keysafe5-install/demoCA/demoCA.conf \
      -out ${x509user}.crt -in ${x509user}.csr -batch
    rm ${x509user}.csr
  done
$ kubectl create namespace nshieldkeysafe5
$ kubectl create secret generic ks5-amqptls \
  --namespace nshieldkeysafe5 \
  --from-file=ca.crt \
  --from-file=tls.crt=ks5.crt \
  --from-file=tls.key=ks5.key
$ ${RUN_RABBIT} /opt/bitnami/rabbitmq/sbin/rabbitmq-plugins enable \
  rabbitmq_auth_mechanism_ssl
$ kubectl port-forward --address 0.0.0.0 --namespace rabbitns \
  svc/rabbit-chart-rabbitmq 5671:5671 > k8s-rabbitmq-portfoward.log 2>&1 &
```

Note that we need to keep a set of AMQP TLS credentials for the KeySafe 5 agent.

```
$ tar -zcf ~/keysafe5-install/agent-amqptls.tar.gz \
  keysafe5-agent.key keysafe5-agent.crt ca.crt
```

### 7.3.3. MongoDB

We recommend that you use your standard secure MongoDB Replica Set installation. This is just an example, and not production-ready.

```
$ kubectl create namespace mongons
$ helm install mongo-chart \
  --set image.tag=4.4.12-debian-10-r38 \
  --set architecture=replicaset \
  --set auth.enabled=true \
  --set auth.username=dummyuser \
  --set auth.password=dummypassword \
  --set auth.database=authdb \
  --set tls.enabled=true \
```

```
  --namespace=mongons \
  bitnami/mongodb --version 11.1.10
```

There will be a message listing the MongoDB server addresses. We save them to a variable for use later.

```
$ export MONGO1=mongo-chart-mongodb-0.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGO2=mongo-chart-mongodb-1.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGODB=${MONGO1},${MONGO2}
```

We then pick up the secrets in the MongoDB configuration

```
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.client-pem}" | base64 --decode | \
  openssl pkey -out mongo-client-key.pem
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.client-pem}" | base64 --decode | \
  openssl x509 -out mongo-client-cert.pem
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.mongodb-ca-cert}" | base64 --decode > mongo-ca-cert.pem
```

We add those secrets in a format that KeySafe 5 can accept.

```
$ kubectl create secret generic mongodb-client-tls \
  --namespace=nshieldkeysafe5 \
  --from-file=ca.crt=mongo-ca-cert.pem \
  --from-file=tls.crt=mongo-client-cert.pem \
  --from-file=tls.key=mongo-client-key.pem
$ rm mongo-ca-cert.pem mongo-client-key.pem
```

Access the MongoDB shell to create a user with read/write permissions on the `hsm-mgmt-db` and `sw-mgmt-db` collections. Note that the username needs to match the subject of the client certificate, as found by the command:

```
$ openssl x509 -in mongo-client-cert.pem -subject | head -n 1
```

In this example we will use `mongo-chart-mongodb.mongons.svc.cluster.local`

To run the mongo client container:

```
$ export MONGO_RUN="kubectl -n mongons exec mongo-chart-mongodb-0 0 -- "
$ export TLS_PRIVKEY="$(${MONGO_RUN} bash -c 'cat /certs/mongodb.pem')"
$ export TLS_CERT="$(${MONGO_RUN} bash -c 'cat /certs/mongodb-ca-cert')"
$ export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace mongons \
  mongo-chart-mongodb -o jsonpath="{.data.mongodb-root-password}" \
  | base64 --decode)
$ kubectl run --namespace mongons mongo-chart-mongodb-client \
  --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" \
  --env="TLS_PRIVKEY=$TLS_PRIVKEY" --env="TLS_CERT=$TLS_CERT" --env="MONGODB=$MONGODB" \
  --image bitnami/mongodb:4.4.12-debian-10-r38 --command -- bash
```

Once inside the mongo client container, we need to set up a connection to the server before we can start mongo admin and create the user. After we finish creating the user, we need to exit mongo admin, and then the mongo-client container.

```
$ echo "$TLS_CERT" > /tmp/tls.crt
$ echo "$TLS_PRIVKEY" > /tmp/tls.key
$ mongo admin --tls --tlsCAFile /tmp/tls.crt --tlsCertificateKeyFile /tmp/tls.key \
  --host $MONGODB --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

> use $external
> x509_user = {
   "user" : "CN=mongo-chart-mongodb.mongons.svc.cluster.local",
   "roles" : [
     {"role": "readWrite", "db": "hsm-mgmt-db" },
     {"role": "readWrite", "db": "sw-mgmt-db" },
   ]
 }
> db.createUser(x509_user)
> exit
$ exit
```

## 7.4. Install KeySafe 5

```
# Get Ingress IP address
$ export INGRESS_IP=$(kubectl --namespace istio-system get svc -l app=istio-ingressgateway -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')

# Install the KeySafe 5 backend services
$ helm install keysafe5-backend \
  --namespace=nshieldkeysafe5 \
  --set hsm_mgmt.image=$DOCKER/keysafe5/hsm-mgmt:1.0.0 \
  --set sw_mgmt.image=$DOCKER/keysafe5/sw-mgmt:1.0.0 \
  --set database.type=mongo \
  --set database.mongo.hosts="$MONGO1\,$MONGO2" \
  --set database.mongo.replicaSet=rs0 \
  --set database.mongo.auth.type=tls \
  --set database.mongo.auth.authDatabase=authdb \
  --set database.mongo.tls.enabled=true \
  --set database.mongo.tls.existingSecret=mongodb-client-tls \
  --set amqp.URL=${RABBIT_URL}/${RABBIT_VHOST} \
  --set amqp.auth.type=tls \
  --set amqp.tls.enabled=true \
  --set amqp.tls.existingSecret=ks5-amqptls \
  --wait --timeout 10m \
 helm-charts/nshield-keysafe5-backend-1.0.0.tgz

# Install the KeySafe 5 UI
$ helm install keysafe5-ui \
  --namespace=nshieldkeysafe5 \
  --set ui.image=$DOCKER/keysafe5/mgmt-ui:1.0.0 \
  --set svcEndpoint="https://${HOSTNAME}" \
  --set authMethod=none \
  --wait --timeout 10m \
helm-charts/nshield-keysafe5-ui-1.0.0.tgz

# Create the TLS secret for the Istio Ingress Gateway
$ openssl genrsa -out istio.key 4096
$ openssl req -new -key istio.key -out istio.csr \
```

```
  -subj "/CN=${HOSTNAME}" \
  -addext "keyUsage=digitalSignature" \
  -addext "extendedKeyUsage=serverAuth" \
  -addext "subjectAltName=DNS:${HOSTNAME},IP:${INGRESS_IP}"
$ openssl ca -config ~/keysafe5-install/demoCA/demoCA.conf \
  -out istio.crt -in istio.csr -batch
$ kubectl -n istio-system create secret tls \
  keysafe5-server-credential --cert=istio.crt --key=istio.key

# Configure Istio Ingress Gateway for KeySafe 5
$ helm install keysafe5-istio \
  --namespace=nshieldkeysafe5 \
  --set tls.existingSecret=keysafe5-server-credential \
  --set requireAuthn=false \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.0.0.tgz
```

## 7.5. Access KeySafe 5

You can now access KeySafe 5 on either `https://$INGRESS_IP` or `https://$HOSTNAME`. For example, you could send curl requests:

```
$ curl -X GET --cacert demoCA/cacert.pem https://${HOSTNAME}/mgmt/v1/hsms | jq
$ curl -X GET --cacert demoCA/cacert.pem https://${INGRESS_IP}/mgmt/v1/pools | jq
$ curl -X GET --cacert demoCA/cacert.pem https://${HOSTNAME}/mgmt/v1/worlds | jq
```

You can access the Management UI in a web browser at `https://$HOSTNAME`

## 7.6. Configuring nShield client machines

To configure a host machine to be managed and monitored by this deployment, run the KeySafe 5 agent binary on the nShield client machine containing the relevant Security World or HSMs.

Configure this KeySafe 5 agent to communicate with the same AMQP service as the previous deployment.

You will need to copy the agent-amqptls.tar.gz created above.

Ensure no firewall rules are blocking the AMQP port communication between the machine exposing the AMQP port from Kubernetes and the machine running the agent.

```
$ sudo tar -xf keysafe5-install/keysafe5-agent/keysafe5-1.0.0-Linux-keysafe5-agent.tar.gz -C /
$ export KS5CONF=/opt/nfast/keysafe5/conf
$ cat << EOF | sudo tee $KS5CONF/config.yaml
  logging:
    level: info
    format: json
```

```
    file:
      enabled: true
      path: /opt/nfast/log/keysafe5-agent.log

  amqp:
    url: "${INGRESS_IP}:5671/${RABBIT_VHOST}"
    auth_type: "tls"
  update_interval: 15s
  health_interval: 15s
EOF

$ sudo mkdir -p $KS5CONF/amqp/tls
$ sudo tar xf agent-amqptls.tar.gz -C $KS5CONF/amqp/tls
$ sudo mv $KS5CONF/amqp/tls/keysafe5-agent.key $KS5CONF/amqp/tls/tls.key
$ sudo mv $KS5CONF/amqp/tls/keysafe5-agent.crt $KS5CONF/amqp/tls/tls.crt

$ sudo /opt/nfast/sbin/install
```

## 7.7. Uninstall

```
helm --namespace nshieldkeysafe5 uninstall keysafe5-istio
helm --namespace nshieldkeysafe5 uninstall keysafe5-backend
helm --namespace nshieldkeysafe5 uninstall keysafe5-ui
helm --namespace rabbitns uninstall rabbit-chart
helm --namespace mongons uninstall mongo-chart
```

# 8. Troubleshooting

## 8.1. Central platform

To view the KeySafe 5 application service logs, see Obtaining Logs.

If a Kubernetes resource is not working as expected, use `kubectl describe` to display any errors with that resource:

```
$ kubectl describe pod nshield-keysafe5-0
...
81s         Warning   FailedMount        pod/nshield-keysafe5-0         MountVolume.SetUp failed for volume
"keysafe5-amqp-basicauth-volume" : secret "amqp-credentials" not found
```

You can also use `kubectl get events` to detect errors:

```
$ kubectl get events --all-namespaces
```

For more information on debugging Kubernetes applications, see the Kubernetes documentation here.

## 8.2. KeySafe 5 agent

If the agent fails to start, ensure that the configuration file is present at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml`.

If the configuration file is present but the agent still fails to start, see the Logging: KeySafe 5 agent section for instructions on accessing the log.

If you are using TLS, ensure that the private key and certificate files are present in `%NFAST_DATA_HOME%/keysafe5/conf/amqp/tls`.

If you are using TLS authentication, ensure that the username is specified in the X509 field expected by RabbitMQ, and the level of indirection configured on the RabbitMQ server is correct (see RabbitMQ: Certificate Chains and Verification Depth).

# 9. Obtaining Logs

## 9.1. Central platform

The KeySafe 5 application is configured to log to `stdout`. This means you can view logs by running standard `kubectl` commands.

To get the KeySafe 5 backend services logs:

```
$ kubectl logs nshield-keysafe5-0 hsm-mgmt
$ kubectl logs nshield-keysafe5-0 sw-mgmt
```

To get the KeySafe 5 UI logs:

```
$ UI_POD=$(kubectl -n nshieldkeysafe5 get pods -l app=keysafe5-ui-app -o jsonpath='{.items[0].metadata.name}')
$ kubectl logs $UI_POD
```

Because all logs are directed to `stdout`, you can integrate the application logs with third-party log monitoring tools such as Prometheus or Splunk.

## 9.2. KeySafe 5 agent

### 9.2.1. Linux

The KeySafe 5 agent log file is located at `/opt/nfast/log/keysafe5-agent.log`, unless configured otherwise.

### 9.2.2. Windows

The KeySafe 5 agent log file is located at `C:\ProgramData\nCipher\Log Files\KeySafe5-agent.log`, unless configured otherwise.

The KeySafe 5 Windows Service actions are emitted to the Windows event log under the `nShieldKeySafe5` source identifier.

You can use the `nshieldeventlog` utility to extract these log entries and output them to the console or a text file.

```
nshieldeventlog.exe --source=nShieldKeySafe5
```

As required, specify:

- `-c` | `--count`: The number of records read from the event log.

  The default is `10000`

- `-f` | `--file`: The output filename.

See the nShield Security World Software documentation for more information on the `nshieldeventlog` utility.

# 10. Database

All persistent data for KeySafe 5 is stored in the MongoDB database.

## 10.1. Databases

KeySafe 5 stores data in two different databases within MongoDB. One database for storing HSM Management related data, and one database for storing Security World Management related data.

The names of the databases used within MongoDB are defined by the `helm-keysafe5-backend` Helm chart.

| Key | Description | Default value |
|---|---|---|
| hsm_mgmt.dbName | Name the of database to use for storing persistent HSM data | hsm-mgmt-db |
| sw_mgmt.dbName | Name the of database to use for storing persistent Security World data | sw-mgmt-db |

## 10.2. Collections

### 10.2.1. HSM Management database

KeySafe 5 stores nShield HSM data in the following collections:

- hsms
- pools

### 10.2.2. Security World Management database

KeySafe 5 stores nShield Security World data in the following collections:

- worlds

For each Security World known to KeySafe 5, the following collections are automatically created, where each collection name is prefixed by the ID of the Security World database record that the collection corresponds to:

- <id>_actions

- <id>_authorizations
- <id>_authorized_pools
- <id>_cards
- <id>_cardsets
- <id>_domains
- <id>_groups
- <id>_keys
- <id>_module_certs
- <id>_operations
- <id>_p11objects
- <id>_softcards

## 10.3. User Roles

MongoDB has the notion of roles, where each role has a defined set of allowed actions. A user of a MongoDB database can be given a role which then determines what the user can and cannot do to the data.

For details about MongoDB roles, see the MongoDB documentation.

From a security point of view we want to give KeySafe 5 as a user of the MongoDB database the least privileges which suffice for the functionality it requires from the MongoDB database.

The documentation below details the minimum privileges required for a KeySafe 5 MongoDB user for each database created by KeySafe 5.

### 10.3.1. HSM Management database

The following actions are required by KeySafe 5 for the operation of MongoDB for the HSM Management collections:

- find
- insert
- update
- remove
- createIndex

The MongoDB administrator will configure the HSM Management database with

the following actions and privileges for KeySafe 5 hsm-mgmt-db-user role:

```
dbname = hsm-mgmt-db
actions = ["find", "insert", "update", "remove", "createIndex"]
privileges=[
  {"resource" : {"db": dbname, "collection": "hsms"}, "actions": actions},
  {"resource" : {"db": dbname, "collection": "pools"}, "actions": actions},
]
```

### 10.3.2. Security World Management database

As KeySafe 5 creates new collections in the Security World Management Database as new Security Worlds are introduced to the system, RBAC must be applied at the database level rather than individual collections.

The following actions are required by KeySafe 5 for the operation of MongoDB for the Security World Management collections:

- createIndex
- find
- insert
- remove
- update
- dropCollection

The MongoDB administrator will configure the Security World Management database with the following actions and privileges for KeySafe 5 sw-mgmt-db-user role:

```
privileges=[
  {
    "resource": {"db": "sw-mgmt-db", "collection": ""},
    "actions": ["createIndex", "find", "insert", "remove", "update", "dropCollection"]
  },
]
```

## 10.4. Authentication Methods

KeySafe 5 supports the following authentication mechanisms for access to the MongoDB server:

- No authentication
- SCRAM

- X.509 certificate authentication

The type of authentication is specified by `database.mongo.auth.type` value in the `helm-keysafe5-backend` Helm chart.

### 10.4.1. No Authentication

This option is used during development. It should not be used during production.

### 10.4.2. SCRAM

Using Salted Challenge Response Authentication Mechanism (SCRAM), MongoDB verifies the supplied credentials against the MongoDB's username, password and authentication database.

In the `helm-keysafe5-backend` Helm chart:

- `database.mongo.auth.type` must be set to `pwd`
- `database.mongo.auth.existingSecret` must be set to the name of an existing Kubernetes Secret that contains the username and password to use (the Secret must contain a value for `username` and `password` keys).
- `database.mongo.auth.authDatabase` must be set to the name of MongoDB's authentication database.

### 10.4.3. X.509 Certificate Authentication

KeySafe 5 can use X.509 certificates instead of usernames and passwords to authenticate to the MongoDB database.

In the `helm-keysafe5-backend` Helm chart:

- `database.mongo.auth.type` must be set to `tls`
- `database.mongo.tls.enabled` must be set to `true`
- `database.mongo.tls.existingSecret` must be set to the name of an existing Kubernetes Secret that contains the TLS certificates to use (the Secret must contain the keys `tls.crt`, `tls.key` and `ca.crt`).

## 10.5. Backup

To be able to restore the KeySafe 5 application, Entrust recommends that you

regularly backup the MongoDB database as suggested in the MongoDB documentation.

## 10.6. Maintenance

The KeySafe 5 application (`helm-keysafe5-backend` Helm chart) does not support having database collections removed while the application is running. If deleting collections, or replacing the MongoDB server that KeySafe 5 uses, then please stop the `helm-keysafe5-backend` Helm chart before performing database maintenance and restart the application once the database maintenance is complete.

# 11. Base64url Encoding

Base64url encoding is a method of encoding binary data for safe, unambiguous transmission as plain text, for example in URLs. Full details of how to encode and decode are detailed in https://tools.ietf.org/html/rfc4648#section-5. Since standard base64 encoding uses characters that are reserved for other uses in URLs, a variant has been created that uses alternative characters that do not require escaping when forming part of a URL. Padding characters are also removed and can be derived through the length of the base64url-encoded data, as specified in https://tools.ietf.org/html/rfc7515#appendix-C.

The following python script demonstrates how to encode a file to base64url-encoded format:

```
#!/usr/bin/python

import base64
import sys

with open(sys.argv[1], "rb") as input_file:
    in_data = input_file.read()
    out_b64 = base64.urlsafe_b64encode(in_data).strip("=")
    print out_b64
```

The following python script demonstrates how to pad and decode:

```
#!/usr/bin/python

import base64
import sys

pad = {0: "", 2: "==", 3: "="}

with open(sys.argv[1], "rb") as input_file:
    in_b64u_stripped = input_file.read()
    mod4 = (len(in_b64u_stripped)) % 4
    if (mod4 == 1):
        raise ValueError("Invalid input: its length mod 4 == 1")
    b64u = in_b64u_stripped + pad[mod4]
    out_data = base64.urlsafe_b64decode(b64u)
    print(out_data)
```

# 12. Supported TLS Cipher Suites

This appendix and the helm values.yaml file both use the OpenSSL project's identifiers for TLS Cipher Suites.

**Recommended Cipher Suites: The Default List**

The following TLS Cipher Suites are supported by KeySafe 5, and are configured for use by default. It is strongly recommended that this default set of cipher suites, or a subset of it, is used.

- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-CHACHA20-POLY1305
- ECDHE-RSA-CHACHA20-POLY1305

**Less Secure Cipher Suites: Not Recommended**

The following TLS Cipher Suites are supported by KeySafe 5, but only if explicitly configured for use by the user. These are less secure cipher suites and should only be configured for use after a thorough threat analysis of the operating environment.

- ECDHE-RSA-AES256-SHA
- ECDHE-RSA-AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA
- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES256-SHA
- AES128-SHA
- DES-CBC3-SHA

**TLSv1.3 Cipher Suites: Not Configurable**

The following TLS Cipher Suites are supported by KeySafe 5 and cannot be explicitly configured. These are all secure TLSv1.3 cipher suites.

- TLS_AES_256_GCM_SHA384

- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256